THE INTERDATA 74 COMPUTER

This document describes the interdata 74 processor and memory at a level which is appropriate for programming the machine in an assembler level language.

First a very brief description is given of all the registers in the processor over which programs have explicit control. The diagram below shows these. In describing any register a basic characteristic is the number of bits it contains. By convention, the bits within the register are numbered 0,1,2, etc... As can be seen from below the model 74 registers each have 16 bits.

```
 15               0
|_________________| PS        } Program status
|_________________| LOC       } Word

 15               0
|_________________| R0        }
|_________________| R1        }
|_________________| R2        } General Purpose
:                 :           }
:_________________:           } Registers
|_________________| R14       }
|_________________| R15       }
```

The General Purpose Registers

The 16 registers will be denoted by R0,R1, ... R13,R14,R15. Each register is 16 bits long. They are used for 2 purposes viz:

1) accumulators   Arithmetic (add,subtract) and logical (and, shift) operations can be performed on the contents of any one of these registers. More specifically, whenever an instruction being executed by the processor invokes an arithmetic or a logical operation, then at least one of the operands comes from a general purpose register and the result is placed in the same register. All arithmetic in this machine is done using 2s complement representation in 16 bits.

2) Index registers.   When an instruction requires to access memory the contents of any one of the registers R1 through R15 may be used in the calculating the address of the required location.

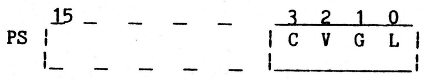
Program Status Word

This is a 32 bit register which can be considered as 2 separate 16 bit registers for most purposes. The complete 32 bit register is denoted by PSW, its most significant 16 bits are referred to as the status part PS asnd the least significant 16 bits are referred to as the location counter, denoted LOC.

```
 15          4 3    0      15             0
|____________|______|     |_______________|
      PS        CC               LOC
                |
         the condition code
```

The significance of these parts is briefly described here. While the processor is running (executing instructions) the location counter (LOC) always contains the address of the instruction currently being executed. Note that an address is a 16 bit quantity. The status part (PS) of the PSW determines the state of the processor, only the 4 least significant bits of PS need to be considered here. They are referred to as the condition code, denoted CC. The bits of the condition code are set to reflect the state of the last operand derived within the processor. They are set by most instructions which place a 16 bit value in a general purposer register and instructions which compare two 16 bit or two 8 bit values. This means that the condition code bits are set at the conclusion of most instructions.

The condition code bits are denoted by C,V,G & L as shown below



Their significance is as follows:-

L	when set to 1 implies that the value was Less than 0 (<0)
G	when set to 1 implies that the value was Greater than 0 (>0)
V	when set to 1 implies that arithmetic overflow occurred during the last arithmetic operation.
C	when set to 1 implies that the last arithmetic or logical operation generated a carry bit.

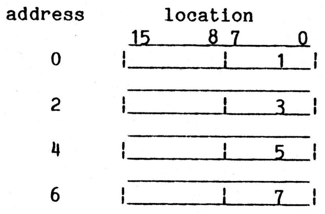
The V,G and L bits are set according to the value interpreted as a 2's complement 16 bit quantity.

The condition code plays a vital role in determining the sequence of instructions selected by the processor.

Main Memory

The main memory of the computer is usually referred to simply as its memory. It is the memory in which all instructions and data reside while a program is being executed.

In describing any main memory it is usual to state the number of bits there are in each location. In this machine, and most others, this is not straightforward. The memory of an interdata 74 can be regarded either as a collection of 16 bit locations or as a collection of 8 bit locations. The diagram below shows how these entities are considered to be laid out, in particular it indicates what address value is given to each location.



etc.......

All addresses generated by programs are 16 bit values. Whether a particular address is referring to a 16 bit or an 8 bit location is determined entirely by the opcode of the instruction being executed.

As implied in the diagram, successive 16 bit locations are addressed by the even integers. In fact if the opcode specifies that a 16 bit location is to be used then the least significant bit of the memory address generated is considered to be 0. Programmers should usually ensure that it is actually 0.

If the opcode implies that an 8 bit location is to be addressed then an even address is taken to refer to the most significant half of the corresponding 16 bit location (bits 15-8), an odd address is taken to refer to the least significant half of the 16 bit location at the immediately preceeding (even) address (bits 7-0).

8 bit locations or values are referred to as "byte" locations or values, whereas 16 bit locations or values are referred to as "half words". (Use of the term half might be thought to imply that there are full word quantities, but these do not exist on this machine.)

The number of locations in the memory can vary from machine to machine. The number is usually expressed in units of a "kilobyte" (Kb), which is 1024 (2**10) bytes. The maximum amount of memory permitted in a machine is 64Kb i.e. addresses in the range 0 to FFFF(hex), but most machines have only 8Kb i.e. addresses in the range 0 to 1FFF.

<u>Processor Instructions</u>

The interdata 74 machine has a large number of different instructions, about 220 different opcodes. Before describing their format and detailed meaning they can all be placed in one of 5 different categories as follows

Load & store	instructions which move values between the general purpose registers and the main memory.
Arithmetic	instructions which add, subtract and divide etc. values.
Logical	instructions which perform operations such as and, or shift etc.
branch	instructions which transfer control such as jumps and subroutine entry instructions.
privileged	instructions which explicitly operate on the program status word or communicate with peripheral devices.

Instruction Formats

There are several formats of instruction. They differ according to the number and significance of the various fields within the format. An instruction will be either 16 or 32 bits long (1 or 2 halfwords). The basic formats are considered in turn in the following sections.

Register - Register (RR) Format

Instructions of this format occupy 1 halfword, there are 3 fields

```
 15__________8_7____4_3____0
|            |      |      |
|  OPCODE    |  Ri  |  R2  |
|____________|______|______|
```

The fields are an 8 bit opcode and two 4 bit fields which specify 2 of the general purpose registers. Opcodes denoting instructions of this format invoke operations on the contents of the 2 registers specified. If a binary operation, such as addition, is called for then the contents of R_i and R_2 are taken as the operands and the result is placed in R_i - overwriting the original operand.

For example, opcode 0A denotes an addition of this format. An instruction such as:-

```
 ___________________________
|__0A________|__4__|__9____|
```

would add the contents of R4 and R9 placing the result in R4.

Register - Indexed Memory (RM) Format

This is a 32 bit format in which there are 4 fields

```
 15__________8_7____4_3____0    15__________________0
|            |      |      |   |                    |
| OPCODE     |  Ri  |  Rx  |   |       ADDR         |
|____________|______|______|   |____________________|
```

An opcode denoting an instruction of this format invokes an operation between the contents of R_i and the contents of a memory location. The address of the memory location is determined by the ADDR and R_x fields. If the R_x field is zero then the address is simply the contents of the ADDR field. However, if R_x is not zero, then the address of the memory location is obtained by summing ADDR and the contents of R_x. A register used in this way to calculate addresses is known as an INDEX REGISTER. The general purpose registers R1 - R15 may be used for this purpose, but not R0.

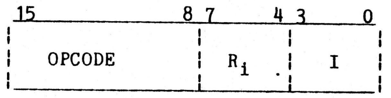
For example, consider the opcode 4A which invokes an addition of this format, thus

```
 ____________________    _____________________
|__4A______|__3_|_E_|   |________1C0_________|
```

The contents of a memory location are added to R3. The address of the location is the sum of ADDR, i.e. 1C0, and the contents of R14.

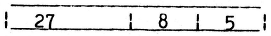
Register - Short Immediate (RSI) Format

This is a 16 bit format having the same division into fields as RR instructions



As before, R_i contains the first of 2 operands and is the destination of the result. In this format the second 4 bit field IS the second operand. The I field is referred to as an IMMEDIATE OPERAND, since it is immediately available within the instruction.

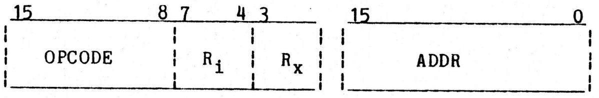
For example, the opcode 27 denotes a subtraction instruction in this format, thus



is an instruction which would subtract 5 from the contents of R8.

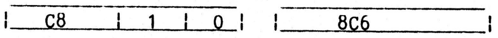
Register - Immmediate (RI) Format

This is a 32 bit instruction which also includes an immediate operand, the field division is identical to RM instructions.



As usual, R_i contains the first operand and is the destination of the result. If the R_x field is zero, then the second operand is the ADDR field. If the R_x field is not zero then the second operand is the sum of ADDR and the contents of R_x. This means that instructions of this format include a full 16 bit immediate operand.

For example, the opcode C8 is a load instruction of this format, it places the 16 bit immediate operand in R_i, thus



places 8C6 into R1.

Other Formats

The four formats defined above cover the vast majority of available instructions. There are, however, formats similar to these in which the R_i field (bits 4-7) does not denote a register, but is used as an immediate operand in some sense. The principal instructions in this form are the branch instructions which are described at the end of the next section.

This section gives a brief summary of most of the instructions in all categories except the privileged category, which will be described later. The various categories of instruction are outlined in turn in the following sections. Some types of operation have opcode variants in all 4 formats, whereas others may have only 2 variants or even a single format. The available formats are indicated simply by giving the opcodes under the four headings RR,RM,RI,RSI. A single mnemonic is used to cover the first 3 formats, but a seperate mnemonic is given for any RSI format.

The tables also indicate the all important effect that the execution of an instruction has on the condition code, the notation is defined in the first section.

Load & Store Instructions

Load instructions place values in general purpose registers, store instructions copy the values in registers usually into memory.

Description	Mnemonic	RR	RM	RI	RSI	mnemonic	CVGL
Load halfword	LH	08	48	C8	24	LIS	00**
Load comp		-	-	-	25	LCS	00**
Load byte	LB	93	D3				----
Exchange byte	EXBR	94					----
Load multiple	LM	-	D1				----
Store halfword	STH	-	40				----
Store byte	STB	92	D2				----
Store Multiple	STM	-	D0				----

The effect on the condition code is defined for each bit as follows

0	imples that this bit is set to zero by the instruction
*	implies that the bit will be set or cleared according to the value
-	implies that the bit is not changed at all by the instruction.

Store operations leave the condition code unchanged. LH and its RSI variants set G or L according to the arithmetic value loaded. Note that the most significant half of R_1 is cleared by the LB instruction. In the RR variant the least significant half of R_2 is loaded to R_1. The EXBR instruction is not strictly a load or store for it takes the contents of R_2 and swops round the most and least significant halves before loading into R_1. The STB instruction in the RR variant places the least significant byte of R_1 in the least significant byte of R_2. The most significant byte of R_2 is unchanged.

The LM and STM instructions transfer values between a number of successive general purpose registers and successive memory locations. The registers involved are all of R_1, R_{1+1}, ... R15 and the memory locations are the one addressed in the instruction and the ones immediately following.

N.B. As their mnemonics suggest, the opcodes determine when byte or halfword addressing is involved.

Arithmetic Instructions

These are the instructions which perform binary operations such as addition and subtraction. The result is almost invariably placed in R_i.

Description	Mnemonic	RR	RM	RI	RSI	mnemonic	CVGL
Add halfword	AH	0A	4A	CA	26	AIS	****
Subtract hw	SH	0B	4B	CB	27	SIS	****
Add hw with carry	ACH	0E	4E				****
Subtract with carry	SCH	0F	4F				****
Add to memory	AHM	-	61				****
Multiply halfword**	MH	0C	4C				----
Mult. hw unsigned**	MHU	9C	DC				----
Divide halfword**	DH	0D	4D				----

** indicates R_i must specify even-odd register pair (see below).

AH and SH are self explanatory. ACH and SCH include the carry bit as an extra least significant bit, which is useful for programming two halfwords as one 32 bit integer (multiple precision arithmetic). AHM is similar to AH except that the result is placed in the memory location NOT in R_i which remains unchanged.

Multiplication of two 16 bit values gives a 32 bit result. For this reason the multiply instructions use not a single register (R_i) but two adjacent registers the first of which is R_i and it must be an even numbered register. These even-odd pairs of registers will be denoted as R_i and R_{i+1}, examples are R0-R1, R4-R5, R14-R15 etc..

Division of two 16 bit values can give a 16 bit quotient or a 16 bit remainder. For this reason the divide instruction also makes use of an even-odd pair of registers. Initially, R_i-R_{i+1} contain the operand which is to be divided, finally the quotient appears in R_{i+1} and the remainder in R_i.

Logical Instructions

These invoke the standard binary logical operations

Description	Mnemonic	RR	RM	RI	RSI	mnemonic	CVGL
And halfword	NH	04	44	C4			00**
Or halfword	OH	06	46	C4			00**
Exclusive or hw	XH	07	47	C7			00**

Shift Instructions

These fall into several sub-groups. Logical shifts move a 16 or 32 bit value as a whole introducing 0's into vacated positions. Arithmetic shifts leave the sign bit (most significant bit) unchanged, they introduce 0's into vacated positions during a left shift but copy the sign bit into the vacated position during a right shift.

Description	Mnemonic	RR	RM	RI	RSI	mnemonic	CVGL
logical							
left 32 bits**	SLL	-	-	ED			*0**
right 32 bits**	SRL	-	-	EC			*0**
left 16 bits	SLHL	-	-	CD	91	SLLS	*0**
right 16 bits	SRHL	-	-	CC	90	SRLS	*0**
Arithmetic							
left 32 bits**	SLA	-	-	EF			*0**
right 32 bits**	SRA	-	-	EE			*0**
left 16 bits	SLHA	-	-	CF			*0**
right 16 bits	SRHA	-	-	CE			*0**
Rotate							
left 32 bits**	RLL	-	-	EB			00**
right 32 bits**	RRL	-	-	EA			00**

Note that all these instructions are of RI or RSI format. The 32 bit operations require that R_i specify the even register of an even-odd pair. Note that rotate instructions do not set the carry bit, whereas all others set it according to the last bit shifted out.

Compare Instructions

These are a very useful group of instructions which do nothing other than set the condition code according to the relative values of the two operands. The CC reflects the relationship between the contents of R_i and the second operand in that order e.g. R_i > 2nd. operand sets the G bit.

Description	Mnemonic	RR	RM	RI	RSI	mnemonic	CVGL
Compare logical	CLH	05	45	C5			*0**
Compare arithmetic	CH	09	49	C9			*0**
Test Halfword	THI	-	-	C3			00**
Compare logical byte	CLB	-	D4				*0**

The compares are all done simply by subtracting the second operand from the value in R_i. R_i is not altered (the result is thrown away!), but the condition code set. The logical comparisons pay no regard to the signs of the operands, whereas the arithmetic comparisons set G and L strictly according to arithmetic comparison. The THI instruction performs a logical AND operation rather than a subtraction but is otherwise similar. CLB compares the least significant 8 bits in a register with an 8 bit value in memory.

Supervisor Call

This is a special instruction of the RI format with mnemonic SVC and opcode E1. The details of the mechanism involved will be given later but the instruction is designed to allow non-privileged programs to request services from some privileged supervisor program(s). The R_i field does not refer to a general purpose register, but designates one of 16 different categories of supervisor service which may be requested. The immediate operand is available to the supervisor program thus invoked. The instruction is like a subroutine call in that after the supervisor has completed the service it has the information to enable it to return to the instruction immediately following the SVC. The effect on the condition code is determined by the supervisor service invoked.

Branch Instructions

These are the instructions which may break the normal sequential flow of instruction execution. They always designate a memory address which contains further instructions. The condition code is interrogated by these instructions, if it conforms to a certain state determined by the instruction then LOC is loaded to cause the next instruction to be taken from the designated memory address, otherwise the instruction immediately following the branch instruction is taken next. In all these instructions the R_i field (4 bits) does not designate a register, but a value against which the condition code (4 bits) is compared. The R_i field is referred to as the "mask" in this context.

Branch instructions are of two kinds. The "branch on true" kind perform a logical AND operation between the mask and the condition code. The branch "is taken" if the result is not zero, i.e. the second operand determines the address of the next instruction. If the result is zero the instruction immmediately following the branch is executed next - the branch is "not taken". The other kind of branch instruction is the "branch on false" kind. Here the mask and condition code are ANDed together, but the branch is taken only if the result is zero.

Description	mnemonic	RR	RM	RI	RSI	nmemonic	CVGL
Branch on true condition	BTC	02	42		20	BTBS	----
					21	BTFS	----
Branch on false condition	BFC	03	43		22	BFBS	----
					23	BFFS	----
Loop control branch index high	BXH	-	C0				----
br. index less or equal	BXLE	-	C1				----
subroutine entry branch and link	BAL	01	41				----

In the RR format BTC and BFC instructions the second register (R_2) contains the address of the next instruction if the branch is taken. In the RM format the address of the second operand is the address of the next instruction if the branch is taken (more like RI form?).

The RSI format instructions determine the branch address from the 4 bit immediate operand which, in this case, denotes a number of halfwords. In the BTFS (branch on true forward short) and BFFS (branch on false forward short) instructions when the branch is taken the next instruction is that which is located the given number of halfwords after (forward from) the branch instruction itself i.e. at LOC + 2*I. If the branch is taken in the BTBS or BFBS (backward short) instructions then the next instruction is located the given number of halfwords preceeding (backwards from) the branch instruction itself i.e. at LOC-2*I.

This paragraph shows a few examples of typical branch instructions. Consider the following BTC instruction

```
 ________________________
|   02   |  3  |   6   |     i.e. BTC 3,R6
```

The mask (3) is ANDed with the CC and the branch taken if the result is not zero. This means the branch is taken if either the G or the L bit is set (or if both - but that is impossible). With the normal arithmetic interpretation this means that the branch is taken if the last result was non-zero (#0 is same as >0 or <0). A branch if the result was equal to zero would be taken by an instruction such as

```
 ___________________
|__03__|__3__|__6__|  i.e. BFC 3,R6
```

Here the mask and CC are anded and the result must be zero for the branch to be taken i.e. G and L must both be zero implying the result was equal to zero. In both instructions the address of the next instruction is contained in R6.

It is possible to make all normal comparisons by noting that, for example <=0 is the inverse of >0.

The diagram below shows the branches that would take place if the following RSI format branches were taken

```
     address            halfword

       n              |____________|
       n+2            |____________|<-
       n+4            |____________| |
       n+6            |____________| |
       n+8  (BFFS)    |____2324____|------      if <=0
      n+10            |____________| |    |
      n+12  (BTBS)    |____2015____|--    |     if <0
      n+14            |____________|      |
      n+16            |____________|<-----
                      |            |
```

The BXH and BXLE are instructions which mimic the action of a high level language construction such as the IMP cycle with a control variable. No condition code testing is involved, instead tests are made on the relative contents of cvarious registers. R_i, R_{i+1} and R_{i+2} play the role of a control variable, an increment and a final value. The action for BXH is to add the contents of R_{i+1} to R_i and compare the new R_i value with the contents of R_{i+2}. The branch is taken if $R_i > R_{i+2}$. Thus three adjacent registers are involved. The BXLE instruction is similar except that the branch is taken only if the incremented R_i value is less than or equal to the contents of R_{i+2}.

Subroutine Entry

This very important facility is provided via the branch asnd link (BAL) instruction. This is an unconditional branch instruction, i.e. it is always taken. The R_i field denotes a register into which the address of the halfword immediately following the BAL instruction is placed when the BAL instruction is being executed. This address is used by the subroutine to return to the correct place at the conclusion of the routine.